COMP9016 Assignment #1

(John) Paul Nagle, R00065426

1.1 Building Your World

Task Environment

The 2D world that has been implemented is a fully observable, deterministic (i.e. actions have predictable effect on state), sequential (actions affect future outcomes i.e. the location of the agent changes), and static (i.e. the env does not change) grid of configurable height and depth.

There is one “Winning” block in the grid. If the agent lands on the “Winning” block, they are awarded 100 points, the game is declared Won, and the game is over.

There is one “Penalty” block in the game. If the agent lands on the “Penalty” block, they are penalised 50 points.

There is one Obstacle block in the game on to which the agent cannot move, and must navigate around.

The agent cannot move outside the boundaries of the grid.

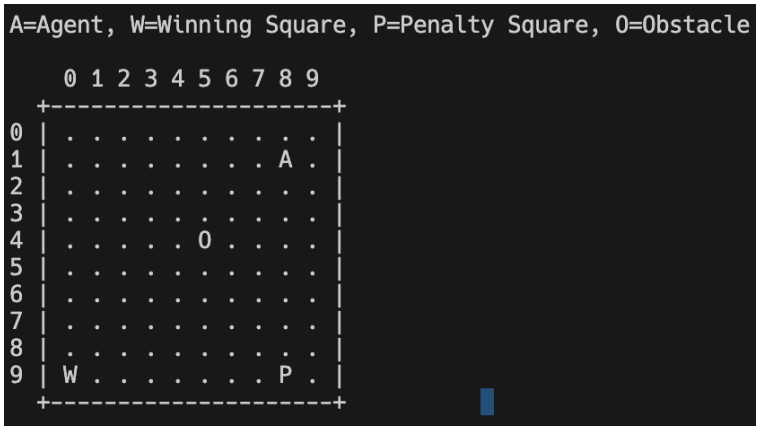
They can move one square per move in any of “up”, “down”, “left” or “right”, provided that the above restrictions are met.

Each move has a cost associated with it, which is the sum of the x and y coordinates of the square.

The game is considered Lost if the agent has not found the Winning block in the number of steps allowed.

When the game starts, the all “Things” are positioned at random locations.

i.e. 2D world with (w=10 d=10) grid



Agent Types and PEAS Descriptions

	Performance	Environment	Actuators	Sensors
Random	+100 on winning block and Win game, -50 on penalty block, -(x+y) per move, Lose game if Winning block not found in S steps	2D grid of width w and depth d, with winning block, penalty block and obstacle block.	Move (Up, Down, Left, Right)	Current position. Percepts are ignored.
Reflex	As above	As above	As above	Current position, receives percepts with available directions and associated costs
Table	As above	As above	As above	As per Reflex Agent. Decides movement based on internal table.

Advantages/Disadvantages of the different agents

	Advantages	Disadvantages
Random	Very fast Easy implementation Effective in small grids	non-rational Non optimised Scales badly
Reflex	Low computational overhead Rational Performs better than Random	Poor performance in large grids Inconsistent performance Non optimised
Table	Makes informed decisions Performs better in larger grids that the other agents	Requires computational overhead Optimal implementation is difficult, how to decide how to build the table? No proper Search or Goal Based decisions

Ability to perform

Random (random_move)	This produces very erratic results, and is impossible to characterise in any rational way. The more steps this agent is allowed, the higher likelihood it has of winning. There was nearly always a non-zero win rate for this agent.
Reflex (cheapest_move)	This agent always tends to move to (0,0) and oscillate between it and (1,0), as these are the cheapest moves in this game. If either happen to be a penalty square, the cost can be huge. If the Winning square is not between the original location of the agent and (0,0) then this agent will never win. 0% win rates were common for this agent.
Table (table_action)	This agent moves in a predictable fashion, but it is down to chance as to whether the Winning square is within the set list of moves that this agent will make. 0% win rates were common for this agent.

Suitability to operate in worlds of varying sizes

Sample results after experiments with 100 different worlds:

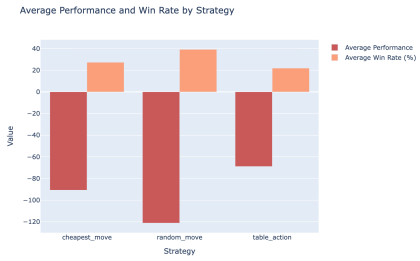


fig. 1 *100 x (STEPS=> 40 RUNS=> 500 WIDTH=> 6 DEPTH=> 6)*

fig. 2 *100 x (STEPS=> 40 RUNS=> 500 WIDTH=> 10 DEPTH=> 10)*

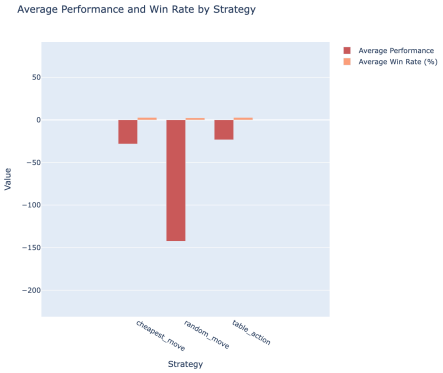


fig 3. 100 x (*STEPS=> 40 RUNS=> 500 WIDTH=> 20 DEPTH=> 20)*

Random	Performance and win rate fluctuated randomly across worlds of different sizes.
Reflex	Win rate was better in smaller worlds, as there was more likelihood of the winning square being on the agent's path to (0, 0). The reflex trigger is the main driver of the poor result, something that didn't always tend to (0,0) might have proved more performant.
Table	Performance improved in the bigger worlds as there was less likelihood of the pre-defined moves hitting the Penalty square. Conversely, Win rate was better in the smaller worlds, as there was more likelihood of hitting the Winning square in the more limited environment. This could have been improved by a better strategy in the table of defined moves. Defining an expanding circular motion, or a systematic row by row search might have improved the outcomes.

1.2 Searching Your World

Problem Formulation

Formulate a well defined problem statement and identify a goal-state under which your game is complete. Why is this important to search? As part of your solution you should be including the initial state, the set of actions, the transition model, a goal test function and a path cost function.

Uninformed Search Techniques

Select three uninformed search techniques and discuss their appropriateness to your world under appropriate headings for evaluating problem-solving performance. Implement the uninformed search techniques and discuss the results

Informed Search Techniques

Select three informed search techniques and discuss their appropriateness to your world under appropriate headings for evaluating problem-solving performance. Implement the informed search techniques and discuss the results.

Performance Evaluation

Write a clear and concise report detailing the search techniques performance in your agent-based game for the relevant agent types. The purpose of this is to articulate an understanding of the underlying concepts, and limitations, being implemented both from a theoretical and practical perspective.

Conclusion

- Summarize key findings
- Reflect on agent suitability and search efficiency
- Suggest improvements or future work